SAND2014-15795R

White Paper:

Proposing an Abstracted Interface and Protocol for Computer Systems

David R. Resnick, Sandia National Laboratories[1]
Michael Ignatowski, AMD Research

Version 1.0 July 7,2014

Motivation for a New Protocol

While it made sense for historical reasons to develop different interfaces and protocols for memory channels, CPU to CPU interactions, and I/O devices, ongoing developments in the computer industry are leading to more converged requirements and physical implementations for these interconnects. As it becomes increasingly common for advanced components to contain a variety of computational devices as well as memory, the distinction between processors, memory, accelerators, and I/O devices becomes increasingly blurred. As a result, the interface requirements among such components are converging.

There is also a wide range of new disruptive technologies that will impact the computer market in the coming years, including 3D integration and emerging NVRAM memory. Optimal exploitation of these technologies cannot be done with the existing memory, storage, and I/O interface standards.

The computer industry has historically made major advances when industry players have been able to add innovation behind a standard interface. The standard interface provides a large market for their products and enables relatively quick and widespread adoption. To enable a new wave of innovation in the form of advanced memory products and accelerators, we need a new standard interface explicitly designed to provide both the performance and flexibility to support new system integration solutions.

What a New Protocol Should Support

A single converged interconnect may well support the following:

- Serve as a memory interface for a wide variety of emerging memory technologies with varying timing requirements and functions.
- Enable intelligent memory operations (Processing-in-Memory) ranging from simple functions like atomic operations to more general processing capabilities.

[1] Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy?s National Nuclear Security Administration under contract DE-AC04-94AL85000.

- Serve as a standard interconnect between processor chips, between processors and accelerators, or between processors and reconfigurable computing elements.
- Serve as a standard interconnect between processors and intelligent I/O devices.
- Support a reasonably general ability to interconnect a modest number of components in a local network to provide significant configuration flexibility.
- Support emerging programming models, and application development environments, such as HSA (Heterogeneous Systems Architecture).
- Support standard approaches to power monitoring and management.
- Support standard approaches to error detection and handling, including built-in-self-test and self-repair.
- A layered protocol with potentially one or more physical layer definitions, a data link/packet layer, a routing layer, and a protocol/command layer.

There are some clear overlaps between the above requirements.

The capabilities mentioned above are expected to have tremendous impact on high-end systems, but will have beneficial impacts on systems at all levels, even a single CPU chip that has only a single 3D DRAM and a single non-volatile memory part.

What the New Protocol Should Not Attempt To Do

- Be a totally universal interface including interfaces to memory. There will continue to be separate low-cost low-function interfaces that will be cost effective for many uses. There will be interfaces, like InfiniBand, that serve very well. At the same time, it is also proposed that the abstract protocol be defined to enable support of at least a few existing interfaces by enabling multiple data formats and protocols to be carried within the implementation of this protocol.
- Define the low level physical interface and signaling requirements. That is a separate exercise. At the same time, the protocol must keep in mind the ranges, limits, and 'druthers of possible physical instantiations.

Motivations for Processor and Component Companies

A wide variety of emerging memory technologies will be introduced the coming years, each with its own unique timing requirements, interfaces, protocols, etc. Supporting a wide variety of memory standards in a rapidly changing memory market is impractical with multi-year processor chip development cycles. In order to exploit emerging memory technologies, the industry must move away from interfaces, e.g., DDR3, that are highly tuned to specific parameters of any one technology and one implementation of that technology.

There are wide ranges of bandwidths and similar parameters that must be accommodated in different system implementations, but that does not mean that the number of options in implementations should be nearly infinite. A single high-level interface protocol can be designed that covers most all interface requirements for bandwidths, limits on pins or packages, etc. with a limited number of options, but implemented in a manner that the same commands, data formats, error detection/correction options (etc.) can be supported within the protocol standard. With this standard interface, a great deal of design complexity can be eliminated and things like error detection/correction can be the same no matter if the interface is driving a set of DRAMs or connecting two CPU chips. And a standard can be developed such that error detection and recovery can be the same even if some of the implementation specifics are different like a different order or the presence/absence of fields in a communication packet.

Development costs would be significantly reduced if a single physical interface (PHY) standard that allows for a limited number of options for data widths and rates, but that covers a fairly wide range of such parameters. Low, mid, and high speed physical interfaces (PHYs) could be agreed on for interconnecting multiple different types of components while keeping the same packet formats and protocols including the same basic commands. This would also enable greater flexibility in the configurations supported by processor chips if the chip I/O PHYs could be more general purpose instead of being designated as memory PHYs, I/O device PHYs, and processor-to-processor PHYs.

Component manufacturers would like to provide value-added innovation under a widely supported standard. Such a standard would have to be flexible enough to support intelligent devices with suitable programming models.

In order to support a wide variety of intelligent components, hardware support that would enable a standard programming model would greatly simplify an otherwise difficult and chaotic software ecosystem. HSA (Heterogeneous System Architecture) is a widely supported emerging open standard programming model that could possibly serve as the base for such a standard. It defines a programming model both in terms of how memory is shared, and how different compute elements interact.

Impact of 3D Memory Stacking

With 3D packaging technology coming on board, as seen in Micron's Hybrid Memory Cubes (HMC) and in High Bandwidth Memory (HBM, a JEDEC DRAM standard), along with no effort to create a DDR5 standard, the memory system and memory components will undergo great changes. The stack concept allows much higher chip densities, lower power and the ability to put new functionality into new components. This offers the opportunity to upgrade memory (and hopefully other interfaces) as part of the inevitable changes in system implementations while keeping the same interface and protocol because they are designed to be upgradable.

One option to take advantage of new capabilities is to put additional logic functions in the new components such that the resultant system runs much more efficiently, while supporting growth into the future, along with a new level of flexibility in architecture, design, implementation and upgradability of components. We have the potential to design a new generation of *intelligent* components that enable development of a wider range of systems with reduced complexity in system integration, use, and support.

Intelligent Memory Components

High-end computer systems keep growing in size, complexity, and power usage.

At the same time, the average activity level in CPU chips is not very high. This is due to multiple causes, most all of which relate to latency. A considerable portion is due to data movement but is also the result of things like poor cache use, data access times, and communication overhead and interaction delays. Rather than waiting for memory to return data that a CPU can then use, an intelligent memory system could operate on the data to achieve the wanted result, or manipulate the data to make the CPU's data use more efficient, e.g., by rearranging data so that CPU caches see increased hit rates.

A major way to both increase performance and reduce energy is to reduce data movement. If an operation can be done directly in the memory system rather than by moving data to and from CPUs, then system performance will increase and energy usage goes down at the same time. What happens to system power is more complex; the memory power goes up for the added operation(s) while the power it takes to interface between a CPU and its memory is reduced because of eliminated activity, along with saved time and effort in the CPU. System energy is reduced because performance goes up, resulting in decreased run times and because the total amount of work is reduced.

Abstracted Interfaces

As technologies become more complex it is becoming increasing harder to integrate new types of components into systems. There is a great need, particularly in HPC, for non-volatile (NV) memory to become an integrated part of memory systems. For such uses, a NAND chip should not function as a disk drive, rather it should be sent read and write commands. And DRAM, which is the great majority of current memory, should similarly get read and write commands rather than RAS/Activate, CAS/Read, Precharge as three commands with specific timing requiring moderately complex control logic in the CPU to execute the needed timing while working to maximize bandwidths.

What is needed, at least with respect to component interfaces, is to abstract the interface. For example, rather than send *RAS Bank# row-address/CAS Read column-address/Precharge Bank#* to a memory component with specific times between the commands, send *Read 128 bytes from address B* or *Write 32 bytes to address F*. Design out the need to care about having things be exactly on time, being cognizant of the address map, and planning for support of timing and other changes required when a new generation of components are used in place of the current generation. Send *Read 512 bytes from address R* to a NAND part rather than *Select Head P/ Cylinder Q/Read sector M.* Furthermore, we can use the same data format and interchange protocol as if the request was going to another CPU or to a DRAM as those components will share a common interface protocol with the NAND.

Would like to have a NIC (network interface chip or subcomponent) that could receive a complex command (say to move multiple different memory blocks of different sizes from multiple sources to a local memory that is faster and is closer to the requesting CPU), and be doing this while the CPU is doing other work. This means more system concurrency (so higher performance), lower energy and higher efficiency for the CPU, as the CPU generates the request to its NIC, does other work, and then processes the rearranged data more easily and quickly with less effort and wait time.

With the concept of interface abstraction, we can support new and better ways to do things within computer systems and can enable new generations of components to provide a better and easier path to system design, easier software development, and higher performance with less total hardware as each component is working more efficiently.

Abstraction can work because logic associated with a component takes requests, implements the control and functions needed to have its components execute the request, so that any requestor sees the abstracted interface. Creating this capability is not free, but the benefits will be well worth the effort.

Benefits of Abstracted Interfaces

A *Read* request to a DRAM is the same as a *Read* to an NVRAM part and is the same as a *Read* to another CPU or even to an I/O device.

> There certainly can and will be differences in the detailed contents of packets to components of very different purpose and function. The differences do not require being in the frames of packets and do not affect the rules of the interface protocol. Parameters that need to be present for one component but not another can be included in the 'data' portion of a packet. In addition, each packet frame will have a field that indicates how second level information is placed in each packet. This last capability means that most components should be able to receive multiple varieties of packets; that can optimize communication efficiency.

Allows for vendors to optimize their own designs, including adding new functions, without breaking existing uses.

> While a component design does have to add control logic to a component, it is also the case that the vendor can optimize a design to best use its own capabilities and therefore save logic area or create other ways to take advantage of its unique capabilities—as long as the changes do not break the expectations of a basic component. A 'Read' must still read, even if there are special flavors added or a flavor is not indicated.

The abstracted interface supports asynchrony—can ignore most timing issues. Generate a request without caring exactly when a reply will be returned.

> This simplifies many interfaces, though it does mean that logic is added to some components, for example, to provide time-out detection. This capability also means that different components, that are designed with the same basic capabilities, can be used and that new generations of components can be designed and then used in older systems, knowing that they will function correctly and within acceptable limits. (Of course the power requirements of a new component could be too high, but that is not something that can be addressed in this high level overview.)
>
> Memory vendors might offer parts with different latencies—for different prices—and all the different parts are interchangeable and work in systems without changing anything in the interface, and in configuration or timing parameters.
>
> This is likely the easiest possible example, but it also means that an NV part with different timing altogether can physically be used in place of a DRAM, possibly with some use caveats.

Supports an on-ramp to new kinds of memory components and new functions.

> As noted just above.

Driver software becomes much simpler

> Most of the complexity of a component will be supported within the component. A driver is reduced to things like managing data buffers and data flow, and error detection and recovery. This will make interfaces easier to design and support as well as increasing system resilience.

Communication and data movement becomes more efficient.

- The interface is simplified.
- The protocol has a primary purpose to enable more capability in components, e.g., the ability to transpose a matrix within the memory system so that a using CPU has high hit rates. Another example is building atomic operations into memory; some CPU traffic is eliminated and the function eliminates coherency issues for those operations.
- Errors can be detected and contained locally, simplifying both hardware and software.

> There are other reasons, but the point is that there are multiple worthwhile benefits.

This concept for intelligent components allows for a wide variety of systems to be more easily integrated as the interface and protocol remain consistent. It allows new commands and new system functions to be introduced with consistent rules in systems that previously did not support those new functions. Of course, it would then require the addition of software to take advantage of the new capability.

> This, hopefully, is clear from the discussion above.

New components can be used in the place of older components if the packaging and power constraints enable that.

The logic can provide the capability for components to have built-in self test (BIST) that are optimum for each component. This capability needs very little support to use (at least if used as go/no-go functionality).

> May well be worthwhile in some components, like 3D-stacked memory parts, to include BIST. This test capability can be optimized for each part with none of the differences visible outside the part. This makes debugging easier and more reliable and if, for example, the part also has some self-repair capability that the repair operation was done successfully, with little effort needed by the system to do anything detailed or complex.

Functions can be added in the memory system that make it more capable in direct support of application needs and provide functions that increase system efficiency and performance.

- Atomic operations, already mentioned, to reduce coherency and communication overheads
- Moves (gather/scatter, shuffles, matrix operations like transpose, …)
- Communication accelerators and functions that reduce or replace current CPU functions like providing data base and search functionality directly in the memory system.
- Some CPU-like functionality to test data and do different operations as a result, but within the memory system.

All of the indicated operations mean that additional functions are built into the memory system and that the interface and protocol must be able to support the added functionality.

Again, the idea is to enable the memory system to do things concurrently with other system operations and to reduce the total system energy at the same time that system performance is being upgraded. And to do that in a consistent way so that old and new functions can coexist and that things are defined so that the path for future growth is considered up front, not patched on.

Functions can be put into network interfaces and directly in the system network that provide new functionality and enable increased reliability and system resilience. An example is to communicate alternate paths and recovery steps to take on network errors and failures. A network structure could determine its environment at startup or after a failure, use the returned information to reconfigure itself, and then send status including the failure and the recovered capability. And this could be different each time the network structure is powered up.

There can be new and additional error detection and error recovery capabilities built into components and into how a system is used. The high level of interaction allows components to have the capability to do self-repair without affecting ongoing traffic, for example. Different kinds of memory components (DRAM and NV flash, for example) can detect and repair themselves in very different ways, all without complex software support in the interface or channel driver. [Of course any error should be reported in some fashion so that maintenance can be done if indicated and maintenance protocols followed.]

Abstraction can allow flexibility in configuration and system structure past that envisioned by current channels. An example might be to allow components to chain together such that a memory channel can have anywhere from a single memory part to dozens or more with a single connector without changes needed in the interface software. Reasonable latency differences because of the chain structure are automatically tolerated, as the protocol is latency independent and the abstraction allows address bits to be used differently in different circumstances.

We are not claiming that it is easy to develop the envisioned capability. While the benefits are very large, there are also component and development costs and status-quo component business interests to be considered.

Drawbacks of Abstracted Interfaces

There may be a loss in performance in some cases. Today a processor or GPU memory controller can be highly tuned to achieve maximum performance with a specific DRAM standard. Processor and GPU companies have invested in considerable IP to optimize the order of memory requests for maximum performance. The optimizations possible within a memory module are unlikely to be as robust as those within a high performance memory controller on a processor or GPU. However, it's not clear how big this impact is for most cases. Both Micron HMC and JEDEC HBM parts have greatly increased parallelism with respect to current memory implementations. They have more internal banks and a greater number of memory links that reduce the need for highly optimized DRAM controllers. Further, such memory controller designs tend to be highly optimized only for a rather limited set of memory options compared to the much broader range of memory technologies supported by this new proposed standard.

Interface timing is not fixed. This has both upsides and downsides. A downside is that each requestor may need to buffer additional data. A wider range of latency can complicate some application

optimization. On the other hand, each vendor of a component can make a somewhat different version and not need to provide all the low level details; the CPU or other component that interfaces to a component does not need to support the detail and thus is simpler in the interface logic and generally in the software that generates or uses the data that passes between the components.

Where there are multiple requestors of a component (even multiple cores in a single CPU chip), the response order is not automatically fixed per the order of requests. If this is an issue there could be parameters that place limits on ordering or other mechanisms. It is also true that increasing parallelism brings up the issue independently of each particular interface and its capabilities.

A component with new capabilities needs upgraded software if the new functions are used, though the previous functionality may still work with the older software. The software might be a new or revised library, or might mean that the application needs some refactoring, along a range of possibilities. This might mean that new functionality is added gradually or possibly not at all; but in any case backward compatibility should be preserved. (And yes there are end-cases here. If a change in a component does something like return a result earlier than a previous component, then this could expose a software bug that was not seen before, for example.)

Expect that the ability to do power reduction things like power down a link and then power it back up may possibly cause use issues. Different channels and an interface that could be either electrical or optical (at least as seen by apps and the OS) will have different times to power up and down and to do characterization and training for the different implementations. The upside is that such changes should not affect general operation and that if there are such effects this can be handled in multiple ways.

Components that have been 'dumb' to this point, like DRAM, will be more complex and somewhat higher in cost. The other side of this is that cost is taken from other areas (including things like software drivers and logic interfacing the external components, in addition to an expected reduction in the total hardware in a system because of increased system efficiency) and system performance is increased in multiple ways at the same time that system energy costs are reduced.

A Packetized Approach

None of what is below is cast in stone. These are simply preliminary thoughts.

The main idea is to establish a high level protocol built around packets, how information is passed in the packets, error handing, etc., such that most any component can interact with any other component as long as the two components understand, at least to some level, what each side can ask of the other side. Components, very likely, should be able to return a type and variation code and/or an abbreviated list of their capabilities. Most all information and data that moves between components, for example between a CPU and a memory part is packetized.

As envisioned, this means that each memory part has the capability to interact with other parts of a computer system if those parts have logic to support that interaction. Thus a request from a CPU might go to an initial memory part that then passes the request to another memory part if the first does not have the needed data. The reduction in total effort is considerable:

> Multiple requests into the memory system, data back and forth to the CPU a couple of times, code running in the CPU to test each reference attempt

is replaced with:

> CPU request to first part, direct request from the first to a second part, reply to the CPU

as the CPU will not have to serve as the focal point. Energy is reduced and time is saved.

Packets will have function variations, but all will be variable in length. Would envision the variations would be to support different numbers and sizes of control parameters for component operations with little to no impact on packet framing and no impact on the interface protocol.

At system startup a component can be asked for its capabilities, starting with something like the number of lanes in the link and the data rate for highest bandwidths. The capability can also serve to lower the interface capability to reduce power or as part of error recovery.

The interfaces initialize in a minimum mode (like one lane and lowest data rate) that is then used to establish the running parameters on each side. So if the interface is 8 lanes wide and supports a higher data rate than the base rate, the startup establishes that and switches quickly. All interfaces are also envisioned to be full duplex as it is expected that each side of a link can originate messages and requests.

The protocol will have resiliency features, likely including CRC with variable retry or multiple-bit ECC. (Very much favor using CRC w/auto-retry though there are tradeoffs.) Expect the protocol to be optimized in support of local error recovery to reduce failure cascades. Could envision a protocol with optional multiple levels.

Support of a packet variation might be such that a 'full' format that has wide fields for things like system addresses is used to establish a set of baseline address ranges and then a packet format that has a tag and set of offsets from the address bases is used in place of the original packet variation. This can reduce packet overhead and make communication more efficient.

Other ideas that can further increase the capabilities and efficiencies should easily be possible.

Once the basic protocol features are defined, then we will likely define a set of hardware implementations that all support the protocol, but with different bandwidths and/or data rates.

Would like to aim at an interface definition that supports multiple lane widths for electrical implementations and also optical interfaces. Would hope for something like three or four different data rates (with room for faster rates in the future), so that, for example, both single-sided lower rate and differential higher rates would be supported. Would also expect that there will be a least a few different channel widths, in support of different bandwidths, packaging options, and power levels.

Would expect to drive to a definition that allows any component that supports the protocol to interact with any other component that supports the protocol with at least the basic capabilities defined in the protocol, though probably not at the maximum bandwidth supported by one component or the other.

Given that components that support this interface and protocol are expected to be fairly intelligent (or at least capable of being intelligent) there likely should be direct support for sending multiple parameters and possibly even lists of commands between components. An example of this is a memory part that sees that it does not have the data requested, forwards the request to another memory part, while indicating that the result of the request be returned to the original requestor.

Would hope that the protocol is defined such that some of commands that are sent in the packet frame indicate that the actual commands are contained in the data portion of the packet. This enables for commands and the packet structure to be more general than the limited set of functions that are contained in packet frames. It also enables things like different levels of error checking and failure recovery to be done if the extended capability is provided by the communication components and also for new functions to be included in upgrades and over time.

Contrasting an Abstracted Interface and Protocol with Some Existing Interfaces

Some existing interfaces are described below for comparison. It is possible that the new interface will directly support coherent operations, unlike PCIe and InfiniBand. Only simple software drivers should be needed. Like most protocols, it will be a layered protocol with one or more physical layer definitions, a data link layer, and a transaction layer. The physical layer definition hopefully supports very energy efficient operations for short links, and expands to include definitions that support longer link distances, possibly of distances of around 10 meters (obviously details TBD). The latency tolerance of the abstract interface helps here.

Some Existing Interfaces for Comparison

QPI:

QPI (Intel QuickPath Interconnect) is a device interconnect developed by Intel as a replacement for its Front-Side Bus (FSB). QPI can be used to interconnect between the processor and to an IO hub or network hubs. It supports 20 lanes in each direction at data rates up to 8 GHz (4 GHz with two transfers per clock cycle). Versions are also used on-chip to communicate between the cores and the "uncore" components.

The bus operates on an 80 bit flit, and includes physical, link, routing, protocol layers. QPI dynamically degrades when hard failures occur.

QPI is a proprietary standard defined by Intel.

HyperTransport:

HyperTransport (HT) is a device interconnect developed by AMD. It supports bus widths from 2 to 32b, is full duplex at speeds up to 3.2 GHz with two data transfers per clock cycle.

HT is packet based, and can also be used in a router network.

The HyperTransport standard is controlled by the HyperTransport Consortium. AMD uses proprietary extensions called Coherent HyperTransport to interconnect processor chips.

PCIe

PCIe (also known as PCI Express, or Peripheral Component Interconnect Express) is a widely used standard interconnect for I/O devices. It falls in the middle between a device interconnect and a routed network bus.

PCIe is hot pluggable, and supports auto-configure (sometimes including ROM device drivers). The common PCI-requests include: configuration read/write, I/O read/write, memory read/write, and interrupts. PCIe supports reliable delivery of packets with retry (CRC & acknowledgements).

PCIe supports 1 to 32 duplex lanes, which can be dynamically downsized. The current PCIe 3.0 standard supports speeds up to 8.0 Gtransfers/s. The spec for PCIe 4.0 is expected in 2014 – 15 and will support speeds of 16GT/s.

PCIe is a layered protocol, consisting of a transaction layer, a data link layer, and a physical layer.

PCIe is an open standard defined by the PCI-SIG (Peripheral Component Interconnect Special Interest Group), an industry consortium with over 800 members.

PCI-SIG I/O Virtualization (IOV) Specifications allow multiple operating systems running simultaneously within a single computer to natively share PCIe devices. SR-IOV (Single Root IOV) provides native I/O Virtualization in existing PCI Express topologies where there is a single root complex. MR-IOV (Multi-Root IOV) builds on the SR-IOV specification to provide native I/O Virtualization in new topologies (such as blade servers) where multiple root complexes share a PCI Express hierarchy.

IBM has developed proprietary CAPI (Coherent Accelerator Processor Interface) extensions to encapsulate coherent operations over PCIe 3.0 links.

InfiniBand

InfiniBand is a scalable routed network bus. It supports Quality of Service, failover, virtual channels, remote DMA, variable sized messages, channel send and receive, multicast, and atomic operations. Communication is controlled by APIs supporting basic verbs. InfiniBand supports the following full duplex lane widths: 1x, 4x, 12x. The lane speeds are: FDR= 13Gb/s, EDR = 25Gb/s, and HDF = 50Gb/s (2017).

The InfiniBand standard is controlled by the InfiniBand Trade Association industry consortium.

NVLink

NVIDIA has recently (March 2014) announced a proprietary alternative to PCIe for providing higher speed interconnection between GPUs and processors. Link lengths are restricted compared to PCIe, allowing for more power efficient implementations. Detailed specifications are not known at this time, but initial products are expected in 2016. IBM is expected to support NVLink in its next generation of Power processor chips.

Summary Table:

	DDR4	HMC 1.0	HBM 1.0	QPI	HT 3.1	PCIe 4.0	IB
Data Lane Rate	8.5 Gb/s	15 Gb/s	1Gb/s	9.6 Gb/s	6.4 Gb/s	16 Gb/s	50 Gb/s (HDR)
Data Width	64b	16b x2x4	128b x8	16b x2	2b-32b x2	1b-32b x2	12b x2
Maximum Bandwidth	68 GB/s	240 GB/s	128 GB/s	38 GB/s	51 GB/s	128 GB/s	150 GB/s
Multi-hop Routing	No	Yes	No	Yes	Yes	No	Yes
Error Detection	ECC optional	Internal ECC Packet CRC	ECC optional	Packet CRC	Packet CRC	Packet CRC	Packet CRC
Error Recovery	ECC optional	retry	ECC optional	retry	retry	retry	retry
Support for Coherency	No	No	No	Yes	No Yes cHT*	No	No

* Coherent HT (cHT) – and AMD proprietary extension

Please ask questions of and return comments to Dave Resnick of Sandia National Labs and Mike Ignatowski of AMD:

drresni@sandia.gov
Mike.Ignatowski@amd.com